Lab 6 Beginners REXX Programming

Part 1-Working with the XEDIT Environment:

In this lab we set you up to be able to write any type of program you might need. Let's begin by typing in a program from your z/VM console. To type in a program you should use the same editor that you would use for other work. Within this course, we will be using XEDIT, the CMS editor.

1) After you are logged on (first or second level is fine) you can type this command to begin your XEDIT environment.

```
xedit hello exec
```

2) This puts you into the XEDIT environment, where you can begin creating REXX execs. You can input lines by entering an "i" for INPUT MODE, to begin entering REXX commands. When you enter the line shown below, hit Enter twice to exit the input mode.

```
/* HELLO EXEC – A conversation */
say "Hello! What is your name?"
pull who
if who = "" then say "Hello stranger!"
else say "Hello" who
```

To save the file you have just created or updated, type **`file`** on the command line for the changes to be saved. After this, you see the READY prompt in your CMS environment. Now your first program is ready to run.

To run your newly created REXX program, type in the file name. Now, type in **`hello`** to run the HELLO EXEC. After you enter `hello` and press Enter, you are prompted for your name and can see if your program runs correctly. If any errors occur, go into the XEDIT environment and fix them. If you have questions or concerns about this example, please see the REXX Language module or *REXX Reference* guide for further information.

To stop a program at any point in its execution, just enter the CMS immediate command to Halt Interpretation:

```
hi
```

This stops REXX from running the program and returns control to the CMS environment.

Part 2 – Examples for writing your own REXX programs

In the next few sections you will be asked to write your own REXX programs using the control structures mentioned in the module. To complete these examples, you will need the REXX module you just finished and your active z/VM system.

Example 1:

Write a REXX EXEC that accepts as input the scores of three sports teams and finds their relative standings (first, second, and third place). There are several different ways to complete this task. Hint: You might need to use an “ELSE NOP” for each IF-THEN-ELSE, but there are also other ways to accomplish this.

Example 2:

Write the program described above, but use a SELECT instruction instead of IF-THEN-ELSE, since SELECT was designed for multiple choices.

Part 3: Assignment 1:

This REXX EXEC is a Cartoon Phrase program. When you enter a character’s name, the program should display the character’s well-known phrase. For example, Bugs Bunny would say “What’s up, Doc?”. So you need to add new characters and variable names. Use the template below to begin your program. Data, variables, statements, comments and commands must be filled in. Try writing out the REXX program before typing it into the XEDIT environment to aid in error detection. (Fill in the empty lines with the appropriate fields):

```
/* __________________________ */
say “what cartoon character is your favorite?”
pull _______
select
when ______ = “BUGS BUNNY”  then phrase = “What’s up, Doc?”
________________________________________________________
________________________________________________________
     . . .

end
say ______ “says” phrase
```

After you feel that you have sketched out enough information and all control structures and commands are correct, type the program into the XEDIT environment and check to make sure everything is working correctly. If not go back and fix the errors. If you have any questions, you can refer back to the REXX module under the SELECT instruction. (Remember the key steps; write it, run it, and fix it.)

Part 4: Solving Errors and Fixing Them:

This section was designed to help you locate errors displayed by your z/VM console. Below is a program with two errors. Enter the program as it is written below and run it.

```
/* Arithmetic Example */
say "Enter two numbers (between 1 and 100) and press enter."
pull first second
/* This portion will use arithmetic expressions to manipulate the
numbers /
add = first + second
say “When adding the two numbers you get:” add
 sub = first - second
 say "When subtracting the two numbers you get:" sub
 mult = first * second
 say "When multiplying the two numbers you get:" mult
 div = first/seconds
 say "When you divide first by second you get:" div
 say "This is the end of the example."
```

When you run this program, you should soon see the errors pop up. Remember that REXX pinpoints the location and type of any errors that it finds, so just read through the errors and fix them as you come to them. While you are in the process of fixing the errors, answer the following questions:

(Before you fix the first error)
3.1) The last two lines of the error show the return value and the type of error; copy them in the space provided.____
____
____

3.2) What did you do to fix the first error? ____________________________________
____
____

3.3) After running the program again, you discovered another error. Copy it here:____
____
____

3.4) What did you do to fix this error? ______________________________________
____
____

Part 5 – Tracing and Understanding Programs:

This will be the last section in this first REXX lab. This next section shows the code of two programs and your job is to write out their outputs. There is no need to type the programs into the system; this section lets you demonstrate your understanding of REXX syntax and REXX control structures. Just review the REXX programs below and answer the questions without running the program itself.

4.1)
```
/* The Final Score */
teama = 21
teamb =28
say “Team A just scored a touchdown and made the extra p
point.”
teama = teama + 7
say “Team B just made a field goal.”
teamb = teamb + 3
say “Team A just scored another touchdown, but missed the
extra point.”
teama = teama + 6
select
   when teama = teamb
      then say “It was a tie score!”
```

```
         when teama < teamb
            then say “Team B is the winner, by” teamb – teama
“points!!”
       when teama > teamb
            then say “Team A is the winner, by” teama – teamb
“points!!”
         otherwise
            say “Something went wrong!!”
      end
      say “And that’s the game.”
```

Question: What is the output?

4.2)

```
      /* What Day EXEC */
      do until reply = date(weekday)
         say “What day of the week is it?”
         parse pull reply
         if reply \= date(weekday)
         then say “No, wrong day.  Try again.
      end
      say “Correct, today is” reply
```

Let’s say today is **Friday** and the list of user input is as follows:

Monday
Wednesday
Friday

Question: What is the output?

Part 6 – A Small Calculator

Example 3:

This program may be a bit challenging, but remember to look back at your REXX module and the other examples. This is a small calculator program that retrieves data through loops and a series of questions. First, your program should loop until the user wants to quit. Second, you should ask the user for a number to perform arithmetic on. Then, ask the user to enter “addition”, “subtraction”, “multiplication”, “division” or “average”. Your program will retrieve the string of data and convert all characters to uppercase. Use a function to take only the first three letters of the operation. When you have the first three letters, use one of the control structures discussed (IF-ELSE-THEN or SELECT) to check for these characters in uppercase and ask for another number to compute the operation. The AVERAGE and ADDITION operations should be able to accept many numbers, while all other operations accept only two numbers.

Note: You should make sure you do not divide by 0, or errors will occur!